CLAIMS

1. A method for verifying software source code that includes references to program variables, the method comprising:

processing the source code to derive a set of next-state functions representing control flow of the source code;

replacing the references to the program variables in the source code with non-deterministic choices in the next-state functions;

restricting the next-state functions including the non-deterministic choices to produce a finite-state model of the control flow; and

verifying the finite-state model to find an error in the source code.

2. A method according to claim 1, wherein processing the source code comprises extracting a program counter from the source code, and expressing the next-state functions in terms of the program counter.

3. A method according to claim 2, wherein processing the source code further comprises expressing the next-state functions with reference to a stack pointer associated with a stack used in running the code, and wherein replacing the program variables comprises eliminating substantially all the references to the program variables from the next-state functions, leaving the next-state functions dependent on the program counter and on the stack pointer.

4. A method according to claim 3, wherein restricting the next-state functions comprises limiting the stack

pointer to a value no greater than a predetermined maximum.

5. A method according to claim 1, wherein replacing the program variables comprises eliminating the references to the program variables from the next-state functions, so that the finite-state model is substantially independent of data values of the program variables.

6. A method according to claim 1, wherein processing the source code further comprises expressing the next-state functions with reference to a stack used in running the code, and wherein restricting the next-state functions comprises limiting the stack to a depth no greater than a predetermined maximum.

7. A method according to claim 6, wherein expressing the next-state functions comprises expressing the next-state functions in terms of a stack pointer associated with the stack, and wherein limiting the stack comprises limiting the stack pointer to a value no greater than the predetermined maximum.

8. A method according to claim 7, wherein expressing the next-state functions in terms of the stack pointer comprises incrementing the stack pointer in response to a function call in the source code, up to the predetermined maximum, and decrementing the stack pointer when the function returns.

9. A method according to claim 1, wherein verifying the finite-state model comprises checking the finite-state model against a specification using a model checker.

10. A method according to claim 9, wherein restricting the next-state functions comprises automatically

producing the model from the source code in a form suitable for processing by the model checker, substantially without human intervention in deriving and restricting the next-state functions or in replacing the references.

11. A method according to claim 9, wherein checking the finite state model comprises checking the model against one or more formulas expressed in terms of temporal logic.

12. A method according to claim 9, wherein checking the finite state model comprises finding a counter-example indicative of the error.

13. Apparatus for verifying software source code that includes references to program variables, the apparatus comprising a program analyzer, which is arranged to process the source code so as to derive a set of next-state functions representing control flow of the source code and to replace the references to the program variables in the source code with non-deterministic choices in the next-state functions, and further to restrict the next-state functions including the non-deterministic choices to produce a finite-state model of the control flow, which can be checked by a model checker to find an error in the source code.

14. Apparatus according to claim 13, wherein the program analyzer is arranged to extract a program counter from the source code, and to express the next-state functions in terms of the program counter.

15. Apparatus according to claim 14, wherein the program analyzer is further arranged to express the next-state

functions with reference to a stack pointer associated with a stack used in running the code, and to eliminate substantially all the references to the program variables from the next-state functions, leaving the next-state functions dependent on the program counter and on the stack pointer.

16. Apparatus according to claim 15, wherein the program analyzer is arranged to limit the stack pointer to a value no greater than a predetermined maximum.

17. Apparatus according to claim 13, wherein the program analyzer is arranged to remove the references to the program variables from the next-state functions, so that the finite-state model is substantially independent of data values of the program variables.

18. Apparatus according to claim 13, wherein the program analyzer is arranged to express the next-state functions with reference to a stack used in running the code, which is limited to a depth no greater than a predetermined maximum.

19. Apparatus according to claim 18, wherein the next-state functions are expressed in terms of a stack pointer associated with the stack, and wherein the stack pointer is limited to a value no greater than the predetermined maximum.

20. Apparatus according to claim 19, wherein in the next-state functions, the stack pointer is incremented in response to a function call in the source code, up to the predetermined maximum, and is decremented when the function returns.

21. Apparatus according to claim 13, and comprising a model checker, which is arranged to check the finite-state model against a specification.

22. Apparatus according to claim 21, wherein the program analyzer is arranged to automatically produce the model from the source code in a form suitable for processing by the model checker, substantially without human intervention in deriving and restricting the next-state functions or in replacing the references.

23. Apparatus according to claim 21, wherein the model checker is arranged to check the model against one or more formulas expressed in terms of temporal logic.

24. Apparatus according to claim 21, wherein the model checker is arranged to find a counter-example indicative of the error.

25. A computer software product for verifying source code that includes references to program variables, the product comprising a computer-readable medium in which program instructions are stored, which instructions, when read by the computer, cause the computer to process the source code so as to derive a set of next-state functions representing control flow of the source code and to replace the references to the program variables in the source code with non-deterministic choices in the next-state functions, and further cause the computer to restrict the next-state functions including the non-deterministic choices to produce a finite-state model of the control flow, which can be checked by a model checker to find an error in the source code.

26. A product according to claim 25, wherein the instructions cause the computer to extract a program counter from the source code, and to express the next-state functions in terms of the program counter.

27. A product according to claim 26, wherein the instructions cause the computer to express the next-state functions with reference to a stack pointer associated with a stack used in running the code, and to eliminate substantially all the references to the program variables from the next-state functions, leaving the next-state functions dependent on the program counter and on the stack pointer.

28. A product according to claim 27, wherein the instructions cause the computer to limit the stack pointer to a value no greater than a predetermined maximum.

29. A product according to claim 25, wherein the instructions cause the computer to remove the references to the program variables from the next-state functions, so that the finite-state model is substantially independent of data values of the program variables.

30. A product according to claim 25, wherein the instructions cause the computer to express the next-state functions with reference to a stack used in running the code, which is limited to a depth no greater than a predetermined maximum.

31. A product according to claim 30, wherein the next-state functions are expressed in terms of a stack pointer associated with the stack, and wherein the stack

pointer is limited to a value no greater than the predetermined maximum.

32. A product according to claim 31, wherein in the next-state functions, the stack pointer is incremented in response to a function call in the source code, up to the predetermined maximum, and is decremented when the function returns.

33. A product according to claim 25, wherein the instructions further cause the computer to check the finite-state model against a specification.

34. A product according to claim 33, wherein the instructions cause the computer to automatically produce the model from the source code in a form suitable for checking against the specification, substantially without human intervention in deriving and restricting the next-state functions or in replacing the references.

35. A product according to claim 33, wherein the instructions cause the computer to check the model against one or more formulas expressed in terms of temporal logic.

36. A product according to claim 33, wherein the instructions cause the computer to find a counter-example indicative of the error.